Collaborative Tagging for Software Reuse

A thesis
submitted in partial fulfillment of the requirements
for the award of degree
of

Master of Engineering
in
Software Engineering

Under the Supervision of
Mr. R.K. Bhatia
Assistant Professor
CSED, TIET, Patiala.

Submitted By
Kapil Bhatia
(Roll No 8043112)

May 2006

Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
(Deemed University), Patiala-147004

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled, **"Collaborative Tagging for Software Reuse",** submitted by me in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision and guidance of Mr. R.K. Bhatia.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other University.

Kapil Bhatia

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Mr. R.K. Bhatia
Assistant Professor
Computer Science & Engineering Department,
Thapar Institute of Engineering & Technology,
Patiala- 147004.

Dr.(Mrs) Seema Bawa
Professor and Head
Computer Sc. & Engg. Department
Thapar Institute of Engg. & Technology
Patiala- 147004

Dr. T. P. Singh
Dean of Academic Affairs
Thapar Institute of Engg. &
Technology
Patiala- 147004

Acknowledgement

I wish to express my deep gratitude to Mr. R.K Bhatia, Assistant Professor, Computer Science and Engineering Department for providing his guidance and support throughout the work.He has always been my friend, mentor and guide and his faith in me resulted in this work.

I am also thankful to Dr. (Mrs.) Seema Bawa, Professor and Head, Computer Science and Engineering Department, for her interest in research work and the motivation she imparts to students.

I would also like to thank all the staff members and my co- research friends Rajiv Bammi and Rocky Goyal who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of the thesis.

Last but not the least I would like to thank the blogging community across the world for answering my questions and resolving the issues.

(Kapil Bhatia)

(8043112)

Abstract

The recent progress in the open source and component based technologies has offered many facilities in the field of software reuse. However, this advancement has also brought the challenge such as designing repositories and developing efficient search mechanism to retrieve software assets. Software reuse is a not a new area of research in software engineering,Most of work is going on in this field has not resulted in wide acceptance of reuse programs in organizations due to technical and human issues. Researchers are trying to invent techniques that increase the efficiency and scalability of software repositories.

This dissertation provides an alternative approach of collaborative tagging to organize software artifacts. We present a software reuse framework for the organization of software artifacts, which can be used as the basis for building software repositories. This model includes representation of software artifacts like code snippets, SRS or test cases in a unique representational path form. This model also contains operations to assign tags (or keywords) to artifacts, which include storing, browsing and searching them. The work has been verified with users of the system and subsequent results have been reported and analyzed. As the work in these kinds of systems is still in nascent stage, we have also represented the limitation, apart from advantages and future directions.

Table of Contents

CANDIDATE'S DECLARATION ..... **I**

ACKNOWLEDGEMENT ..... **II**

ABSTRACT ..... **III**

LIST OF FIGURES ..... **VI**

CHAPTER 1 INTRODUCTION **1**

1.1 MOTIVATION 1
1.2 GOALS AND CONTRIBUTIONS 2
1.3 ORGANIZATION OF DISSERTATION 2

CHAPTER 2 BACKGROUND INFORMATION **3**

2.1 TYPES OF SOFTWARE REUSE 3
2.2 SOURCES OF REUSABLE SOFTWARE COMPONENTS 3
2.3 REUSE APPROACHES 5
2.3.1 Proactive and Reactive Approach *5*
2.3.2 Centralized and Distributed Approach *6*
2.4 PITFALLS OF SOFTWARE REUSE 6
2.4.1 Non-Technical Reasons *6*
2.4.2 Technical Reasons *7*
2.5 MISCONCEPTIONS RELATED TO SOFTWARE REUSE 8
2.6 SOFTWARE COMPONENTS AND COMPONENT BASED DEVELOPMENT 8
2.7 CLASSIFICATION SCHEMES FOR REUSE LIBRARIES 9
2.7.1 Faceted Classification *10*
2.7.2 Formal Classification *10*
2.7.3 Clustering algorithms *11*
2.8 LIMITATION OF TRADITIONAL CLASSIFICATION SCHEMES 12
2.9 FOLKSONOMY-THE NEW APPROACH OF CLASSIFICATION 14
2.9.1 Working of Folksonomy *14*
2.9.2 Advantages of the Folksonomy *15*
2.10 USERRANK AND TAG RANK 16
2.11 MEASURING SEARCH EFFECTIVENESS 17
2.12 URI AND TAG URI STANDARD 17
2.12.1 TAG URI [RFC 4151] *18*

CHAPTER 3 RELATED WORK **19**

3.1 METHODS SUGGESTED TO IMPROVE REUSE 19
3.2 RELATED WORK IN COLLABORATIVE TAGGING 20
3.2.1 Economic Advantages of Collaborative Tagging *21*
3.2.2 Clustering in Collaborative Tagging *21*
3.3 SEARCH SYSTEMS 21
3.3.1 Categorization Browsing in Collaborative Systems *22*
3.3.2 Improved Search in Collaborative Systems *22*
3.3.3 Information Retrieval Methods in Collaborative Tagging *22*
3.4 WORK DONE BY OPEN SOURCE COMMUNITY IN SOFTWARE REPOSITORIES 23
3.5 COLLABORATIVE TAGGING IN ORGANIZATION 23

CHAPTER 4 PROPOSED SYSTEM AND IMPLEMENTATION 25

4.1 PROBLEM STATEMENT 25
4.2 OVERVIEW OF THE PROPOSED SYSTEM 25
4.3 IMPLEMENTATION 26
4.3 Registering at Ksearch 27
4.3.1 Adding Artifacts 27
4.3.2 Searching Assets 30
4.3.3 Browsing 31
4.4 RSS FEEDS 32
4.5 EXPERIMENTS AND RESULTS 33
4.5.1 Setting 33
4.5.2 Actors 33
4.5.3 Events and Procedure 33
4.6 DATA ANALYSIS PROCEDURES 34
CASE 1: Semantic Association 34
CASE 2: Power Law 35
CASE 3: Experience Based User Learning 35
CASE 4: Wrong Tagging 37
CASE 5: Sharing and Security 37
CASE 6: Scalability 38

CHAPTER 5 CONCLUSION AND FUTURE WORK 40

5.1 CONCLUSION 40
5.2 FUTURE WORK 41
5.2.1 Integration of already existing tools with Collaborative Tagging 41
5.2.2 Auto Tagging 41
5.2.3 Hierarchal Multi Labeling 41
5.2.4 Building a tagging Interaction model 42

REFERENCES 43

PAPER/PUBLICATION 49

APPENDIX I 50

List of Figures

Figure	Title	Page Number
2.1	A tag cloud	14
2.2	Precision and Recall	17
4.1	Architecture of K-search	18
4.2	Main page of K-search	21
4.3	Registering at K-search	27
4.4	Adding components	27
4.5	Example of Adding components	29
4.6	Assigning tags and checking privacy	30
4.7	Searching assets in K-search	31
4.8	Distinguish Different assets	31
4.9	Popular tags window	32
4.10	Related tags	32
4.11	Tag cloud(After Data Collection)	34
4.12	Power Law	35
4.13	Searching Binary Tree	36
4.14	Making system Learn	36
4.15	Optimized library	37
4.16	Assigning wrong tag	37
4.17	Add users to team list	38
4.18	Tag Systems Performance	38

Chapter 1 Introduction

Software engineering has been a very expensive endeavor and lack of software reuse has been one of the basic parameters of that high cost.

Even there have been lot of software reuse tools available in market today; reuse of software has been virtually non-existent. Also keyword-based systems suffer from their own limitations, which make designing software repositories a difficult task.

Also the other motivation was that most of the effort in this field has gone into software code modules, snippets or components while reuse of other software artifacts and building a complete system which covers the software development cycle was still missing.

In our study of software reuse and repositories, we realized that non-technical issues like team or individual motivation and approaches for software reuse were as important as technical issues. But there was very little work in understanding the psychology of people who are expected to make use of software reuse.

1.1 Motivation

The approach to design the software repositories has always been top down with the classification scheme decided by some particular set of people who develop the software repository instead of people deciding the way that are convenient to them individually or as a team. Apart from that, reuse at team, individual, project and organization level was not clearly understood and solved in previous approaches when designing software repositories.

To integrate the enterprise software repositories and open source projects as to create a common framework of reuse as open source projects are the largest contributor of reuse, they have to be integrated to the existing repositories.

The other factor was to develop a repository system, which is independent of how the particular software artifact is designed or developed. The cost of designing a reuse system

and of using that system motivates higher management to invest in reuse programmes.So our focus was to build a low cost in terms of development and ease of use system which both management and users feel motivated to take advantage for their reuse process.

1.2 Goals and Contributions

- The goal of this dissertation is to provide a software reuse framework based on collaborative tagging. It is targeted towards programmers and other people who are involved in software development process in order to organize their artifacts in their personal libraries and contributing in building an enterprise based reuse system. In order to achieve this goal - our research was focused on two basic areas: software reuse and Collaborative tagging.
- The primary contribution presented in this dissertation is proposing collaborative tagging for organizing software artifacts. This involved identifying resources that need to be tagged. The secondary contributions are: proposed system can be integrated to already existing reuse frameworks. A reuse system has been proposed keeping individual requirements, team requirements and experience of the people involved in the team.

The research presented in this dissertation is not currently being used by any other system or any other organization. But collaborative tagging is rapidly gaining ground on Internet and many applications are coming up with the application of this concept. Our effort is to present a supplementary approach to solve the issues faced by software reuse industry by proposing an alternative approach.

1.3 Organization of Dissertation

Dissertation has been organized in chapters with chapter 2 explaining the basic concepts regarding collaborative tagging and software reuse. Chapter 3, we discuss the previous work, which has happened in this field, or the related work in software reuse research. In chapter 4, we are proposing the new system, its working and are explaining some experiments, reporting the results and analyzing the work. In chapter 5 of this dissertation, we are giving our conclusions. We also have proposed several possible directions for continued research into the development of collaborative systems in this field.

Chapter 2 Background Information

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. This simple yet powerful vision was introduced in 1968. Software reuse has, however, failed to become a standard software engineering practice [1]. The basic purpose of software reuse practice is to design repositories which help in accurate and efficient way of finding and retrieving software components/artifacts. The success of component-based software requires potential customers reusing from some knowledge space [2]. Because of certain technical, organizational, and psychological software engineering research issues and trends, Software reuse is still evolving as a major research discipline. [3] .Generally reuse is confused with redesign or recoding but there is a difference between them. Reuse is the application of existing software artifacts as it is whereas redesign is the act of altering existing software artifacts. Recoding is the discovery of new software artifacts through construction of software code or system designs. [4]. Metadata is created by dedicated professionals in traditional cataloging systems like libraries [16]. Catalogers create metadata, often in the form of Machine-Readable Cataloging (MARC) records for books and other intellectual creations, and this is the basis of most Online Public Access Catalogs (OPAC) in libraries and other institutions.

2.1 Types of Software Reuse

Software reuse is basically categorized as:

White box reuse provides developers the freedom to modify the code. But code modification is also a key source of problems for potential reuse.

Black box reuse aims to avoid these pitfalls by not allowing the modification of retrieved components.[5]

2.2 Sources of Reusable Software Components

Licensing and ownership becomes confusing when software components are sold in a market. Software Assets can be acquired [6] through any of the following methods:

- With the utility asset library supplied with the operating system,compiler or application development environment (e.g., Microsoft®)
- Through purchase from a company or broker, usually without source (e.g., RogueWave®, ILOG®).
- With source code, from an Open Source community (such as SourceForge®), a university, or IBM® AlphaWorks®.
- From an internal company repository.
- Through "mining" of legacy code or previous versions of products in the product-line.
- Through a swap, barter, or purchase agreement or contract with another internal group.
- Created as part of the regular product or application development process, then packaged and published to a corporate or local repository.
- Proactively defined to meet the needs of new projects, then developed prior to an application development project, or as a sub-project.
- Through the evolution, reengineering, or wrapping of existing assets obtained from another source, in order to be more appropriate to current and future projects.
- Created by a distinct component development and maintenance team that develops components and Web services only for release to others.

Therefore, the mode of acquiring a software asset is very different in today's scenario and this action is based on a framework of decisions:-

1. Corporate level decision, of whether to introduce reuse in the practice of software development.

2. A domain engineering decision, of whether to initiate a domain analysis/domain engineering initiative.

3. An application engineering decision, of whether to introduce reuse practices for a specific development project.

4. A component engineering decision, of whether it is worthwhile to develop a specific component to serve a group of project teams.

All these decisions can be quantified in economic terms, and justified by an economic reason.

2.3 Reuse Approaches

There are different ways or approaches a reuse process is executed in organizations.

2.3.1 Proactive and Reactive Approach

There are two basic investment approaches for software reuse:

Proactive investment for software reuse is like the waterfall approach. This approach tends to require a large initial investment, and returns on investment can only be seen when products are developed and maintained.

Reactive investment is an incremental approach to asset building. One develops reusable assets as reuse opportunities arise while developing products.

The fundamental obstacle preventing spread of controlled vocabularies based reuse systems is that the extent of digital libraries, open sources libraries and the web precedes the ability of any one authority to use traditional methods of metadata creation and indexing. While metadata creation is valuable and indispensable within a particular domain or project, it can be costly to implement and can present significant scaling difficulties [11]. Advances in research of automatic metadata generation applications are increasing [12]. It is found by some researchers that such gains in efficiency, were they to be achieved, would allow information professionals to dedicate their efforts on those resources and intellectually demanding metadata activities (i.e. assigning controlled index terms) [13]. Until such time automatic applications are fully realized, describing or indexing the amount of information available on the web, Intranet or previous repositories will remain beyond the scope of any one authority. Some as a useful way in which the indexing role of the information professional is given to the people therefore considers the emergence of ‘collaborative tagging’. [14]

2.3.2 Centralized and Distributed Approach

There are basically two different process approaches [7] for a reuse programme in an organization:-

The **centralized approach** typically has a separate department or organizational unit that is dedicated to developing, distributing, maintaining and providing training about reusable assets.

With the **distributed approach** a reuse program is implemented collaboratively by projects. But most of the times the projects with same product line fall in this category.

Advantages of distributed approach

- There is less overhead cost, as there is no need to create a separate department.
- Asset development costs are distributed among projects. No large up-front investment is necessary.

Disadvantages of distributed approach

- It may be difficult to coordinate asset development responsibilities if there is no common for the reuse program.
- Even if there is a shared vision among projects, it may not be easy for a given project to provide a component that meets the needs of other projects
- There must be a convincing cost/benefit model to solicit active participation.

2.4 Pitfalls of Software Reuse

There have been many reasons [24] for the lack of progress in software reuse:

2.4.1 Non-Technical Reasons

- The reuse community concentrated its research to technical issue like building repositories, tools for search and retrieval of reusable artifacts. But now non-technical factors like organization, process, business drivers and human involvement are becoming more important.

- Failures in the projects were due to not introducing reuse processes
- Lack of commitment by top management
- Many companies/teams/people are still in wrong belief that using OOP approach for setting up the repository will result in successful reuse program.
- Lack of adequate [25] tools to organize, search, and retrieve reusable modules has impeded reuse
- Value of software reuse refers to whether it is more cost effective, in terms of time, money, or personnel, to reuse software as opposed to developing it from scratch each time it is needed. Since software reuse has its own costs, e.g. location, classification, documentation, and storage, the reuse of software is not always valuable.
- The feasibility of software reuse relates to the ease with which software can be effectively reused. Feasibility is dependent on the facilities provided by a software development environment, and the ease with which reusable software candidates can be identified.

2.4.2 Technical Reasons

In addition to this, technical issues [3] have been observed by authors:

1. Ability to scale the reuse for larger systems
2. How to do formal specification sufficiently to do automation of software reuse repositories
3. Reuse and Domain engineering specification need to be broadly defined .
4. Finding out the link between software resue, domain engineering and corporate policy.
5. Role of software reuse in newer development environment like agile methods
6. To quantitatively estimate the number of potential reuses
7. Reuse at run time-development of self-adaptive software

2.5 Misconceptions Related to Software Reuse

But even with these advantages, software reuse has not taken an integral place in enterprise software engineering policy. There are still many *misconceptions* related with reuse [9]: -

Misconception #1: *Software Reuse is a Technical Problem*-Research work in reuse has been focused on technical issues keeping aside the human issues and motivation for management to introduce and sustain reuse programmes in companies.

Misconception #2: *Artificial Intelligence will solve the Reuse Problem*

Artificial intelligence and latest genetic algorithms are believed to build good software retrieval system.

Misconception # 3: *Reusing Code Results in Huge Increases in Productivity.*

There is no direct correlation between creating an efficient reuse system and increase in productivity, whatsoever except the indirect benefits.

Misconception # 4: There is a myth that Building a repository of components [10] will *motivate people* for reusing components.

Misconception # 5: Generally *overgeneralization* of software components when building software components in order to provide as much as possible functionality in a single component.

2.6 Software components and component based development

Component is the encapsulation of business logic or technical functionality which admits a standard interface [8]. Any platform which supports components offers the possibility of building software applications by taking together software components in a way electronic devices are built up from electronic components. This approach applied to software development is referred to as component based development.

The basic advantages [8] of component-based development are:

1. Software Reuse - Encourages higher level of software reuse.
2. Simplifies Testing - Allows testing to be carried out by first testing each of the components before testing the assembly of components.

3. Simplifies system maintenance and modification - Since each of the components are loosely coupled users are free to upgrade and/or add components as needed without effecting other parts of the system.
4. Higher Quality - Since a component can be built and then continuously improved upon by an expert or organization the quality of a component based application will improve over time.
5. Buy vs. Build -By adopting a component based development approach users have the option of buying off-the-shelf components from third parties rather than developing the same functionality in-house.
6. Greater ROI - Significant savings can be gained through purchasing software components rather than developing the same functionality in-house.
7. Shorten Software Development Cycles - The addition of a given piece of functionality will take days rather than months or even years.

2.7 Classification Schemes for Reuse Libraries

A reuse library [11] consists of a repository for storing reusable assets, a search interface that allows users to search for assets in the repository, a representation method for the assets, and facilities for change management and quality assessment. Success of reuse and domain engineering lies in the ability to predict needed variability in future assets. Industry studies have shown that education is a primary factor in better reuse, yet there had been little systematic study of how best to do reuse education. For proper software reuse, there should be some classification schemes in software libraries to classify and organize software components. Its very interesting to understand the properties of these various classification schemes. In one of the path breaking paper by rupen prieto diaz[26]

1. It must accommodate continually expanding collections, a characteristic of most software organizations.
2. It must support finding components that are similar, not just exact matches.
3. It must support finding functionally equivalent components across domains.

4. It must be very precise and have high descriptive power (both are necessary conditions for classifying and cataloging software).
5. It must be easy to maintain, that is, add, delete, and update the class structure and vocabulary without need to reclassify.
6. It must be easily usable by both the librarian and end user.
7. It must be amenable to automation.

2.7.1 Faceted Classification

Then we looked at some specific classification schemes like **Faceted classification** [27]

Classification in a faceted scheme is straightforward. A faceted scheme may have several facets and each facet may have several terms. A hypothetical faceted scheme has two facets, entities and activities, is introduced here to illustrate the classification process.

{entities} {activities}

systems programming

To classify the title in our example, we select from each facet the term that best describes each of the concepts in the title. "Systems" is selected from the entities facet, and "programming," from the activities facet.

Problems with faceted classification

- Knowledge of domain expert is implicit
- Complicates the automation
- Automation of classification requires reverse engg from source code.(source code analysis)

2.7.2 Formal Classification

Limitations of these classification schemes have always encouraged the use of more methodical or mathematical approach such as formal methods.

The advantages of formal [28] specifications**:** -

- Explicit representation of structure, function and behaviors
- Higher precision beyond faceted approach
- Provides basis for automated reasoning
- Formal classification based on this knowledge logically to verify reusability of software component.

But, as we observed these concepts, I came to the conclusion that, ultimately, rigorous ways like formal methods can improve the organization of components. But, to implement a system with formal methods was too tough and resource intensive.

2.7.3 Clustering algorithms

The process [29] of organizing objects into groups whose members are similar in some way .Two or more objects belong to the same cluster if they are "close" according to a given distance (in this case geometrical distance). This is called distance-based clustering. Another kind of clustering is conceptual clustering: two or more objects belong to the same cluster if this one defines a concept common to all that objects. In other words, objects are grouped according to their fit to descriptive concepts, not according to simple similarity measures.

Then, in particular, I studied some particular clustering algorithms. One of them is Fuzzy c-means clustering Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters, that means it requires 2 pre-defined clusters

But if we have to pre-process software components and for forming software groups without knowing the number of clusters and therefore we use Subtractive clustering. Suppose we don't have a clear idea how many clusters there should be for a given set of data. Subtractive clustering, [30], is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. We also looked at neural network approach of clustering data. SOMs Self-organizing maps (SOMs) [31] are a data visualization technique.

-Humans simply cannot visualize high dimensional data

-The way that SOMs go about organizing themselves is by competing for representation of the samples. Neurons are also allowed to change themselves by learning to become more like samples in hopes of winning the next competition. It is this selection and learning process that makes the weights organize themselves into a map representing similarities.

But, ultimately, we came to the conclusion that Software artifacts based on ontology and keywords has been very complex for building software repositories. These categorization and classification methods have been successful theoretically but failed to understand the human issues regarding software reuse.

2.8 Limitation of Traditional Classification Schemes

Ontology classification is the traditional top down approach of classifying data in a library.Up to this stage, we are looking at the core system of classification which is Ontological classification. This scheme works [32] well in some places, of course but that does not mean that it is the default classification scheme. But there are constraints with this scheme of classification in terms of domain and participants

Domain to be organized

- Small corpus
- Formal categories
- Stable entities
- Restricted entities
- Clear edges

Participants

- Expert catalogers
- Authoritative source of judgment
- Coordinated users
- Expert users

As observed that it’s not always the case in software reuse that we get formal categories or clear edges or expert users. And this may be one of the major reasons the research in this field could not bring big results over time.

So, we can say, that ontology classification is

1. Heavily rule-oriented and structure-oriented
2. An individual or committee drafts up a general guideline for an ontology.
3. Rigorous rules are laid down for creating out that skeleton, to ensure that initial population and subsequent added data are internally consistent
4. The everyday user is not at all as talented in making balanced taxonomic trees compared to an information architect.

Except in a very few cases, users have to learn (and/or navigate/use other tools) the vocabulary to use it, let alone use it effectively.

A common theme in the software reuse[33] literature is that if we can only get the right environment in place--- the right tools, the right generalizations, economic incentives, a ``culture of reuse" - then reuse of software will soar, with consequent improvements in productivity and software quality.

The analysis developed in this paper paints a different picture: the extent to which software reuse can occur is an intrinsic property of a problem domain, and better tools and culture can have only marginal impact on reuse rates if the domain is inherently resistant to reuse. One of the greatest challenges facing people who use large information spaces is to remember and retrieve items that they have previously found and thought to be interesting. Suppose we need every artifact with the word "java" and "binarysearch" in it. We can't ask a cataloguer in advance to say "Well, that's going to be a useful category, we should encode that in advance." Instead, what the cataloguer is going to say is, "java plus binarysearch! Is not possible as we’re not going to optimize the search system for that"

2.9 Folksonomy-The New Approach of Classification

Folksonomies [15] are simply the set of terms that a group of users tagged content with. They are not a predetermined set of classification terms or labels. In simple words, a folksonomy is the complete set of tags—one or two keywords—that users of a shared content management system apply to individual pieces of content in order to group or classify those pieces for retrieval. The use of keywords, or tags, that are explicitly entered by the user for each artifact. Multiple tags allow artifacts to belong to more than one category. That helps to Folksonomy allow users to sort items into multiple categories simultaneously. The folksonomy presents a new opportunity for users to participate in the creation of classification systems apart from using them. The ability to reorient the view by clicking on tags or user names, called pivot browsing, provides a lightweight mechanism to navigate the aggregated bookmark collection.

2.9.1 Working of Folksonomy

In practical terms, a folksonomy [17] is the complete set of tags—one or two keywords—that users of a shared content management system apply to individual pieces of content in order to group or classify those pieces for retrieval. However, folksonomies allow users to sort items into multiple categories simultaneously. Major benefit of the folksonomy is beginning with a blank slate on which the structure of a content space can be allowed to develop through use until patterns emerge. The folksonomy presents a new opportunity for users to participate in the creation of classification systems. Of particular interest are its application to categorizing content in developing fields, and allowing the navigation of content sets by browsing.

Figure -2.1: A tag cloud

An important aspect of a folksonomy is that is comprised of terms in a flat namespace: that is, there is no hierarchy, and no directly specified parent-child or sibling relationships between these terms. There are, however, automatically generated "related" tags, which cluster tags based on common URL/URIs as shown in figure 2.1.

2.9.2 Advantages of the Folksonomy

There are various advantages of folksonomy [18]

1. Difference in working of systems - There is a fundamental difference in the activities of browsing to find interesting content, as opposed to direct searching to find relevant documents in a query. It is similar to the difference between exploring a problem space to formulate questions, as opposed to actually looking for answers to specifically formulated questions.
2. It would require comparisons not to search based information retrieval systems, but to browsing activities using other categorization and classification schemes.
3. A folksonomy represents a fundamental shift in that it is derived not from professionals or component/content creators, but from the users of information and documents used in software development process.
4. Anybody can use the system without any training or previous knowledge to participate immediately. Non-trivial and important metadata are captured through these Folksonomies.
5. Feedback is immediate. As soon as user assigns a tag to an item, he looks at the cluster of items carrying the same tag. If that's not what he expected, he is given incentive to change the tag or add another.
6. Individuals have an incentive to tag their materials with terms that will help them organize their collections in a way that they can find these items later. The organizational scheme that emerges for each individual reflects his or her individual information needs.
7. Tagging is useful for personal recall, or finding again what user has seen before.

Tagging promotes the pleasant and sometimes useful discovery of the unexpected. Libraries and used bookstores know that the collocation of books on a shelf by topic creates interesting cluster of items which are sufficiently different that the information seeker would not have thought to explicitly search for them.

8. Tagging is good for novelty, or checks the "interestingness" of a concept of asset.

2.9.3 Limitations of the Folksonomy

1. Ambiguity of the tags can emerge as users apply the same tag in different ways.
2. There are no explicit systematic guidelines and no scope notes
3. Designed primarily to deal with single words
4. In some instances, multiple words are used together in a single tag, without spaces, i.e., "javamouseclickevent"
5. There is no synonym control in the system. This leads to tags that seemingly have similar intended meanings, like "com," "component," and "DLL" all being used to describe materials related to software components.

Spam tagging, or spagging has several negative effects such as false classification, dilution of meaning, and overabundance of advertising. Online communities are also susceptible to flooding with offensive content.

2.10 UserRank and Tag Rank

UserRank [19]: - Given a ranking of users, we can provide a ranking of any set of entities in the system. For example, the top assets in this system are those tagged by the highest ranked users or the experienced software developers. It does indicate, that this asset has been reviewed by the top users, and that the reviews aught to be taken seriously.

TagRank: - The purpose of ranking tags is to help users find the most relevant tags for a particular entity and facet.Instead, TagRank promotes tags used by the top-ranked users, and provides rankings localized to particular entities and facets.

TagRank is designed to help a community navigate the tail and so tag in a much more meaningful way than simply reusing the globally popular tags.

2.11 Measuring search effectiveness

It is possible to measure [20] how well a search performed with respect to these two parameters.

- RECALL is the ratio of the number of relevant records retrieved to the total number of relevant records in the database. It is usually expressed as a percentage.

  Recall= number of target software components retrieved/number of target software components

- PRECISION is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. It is usually expressed as a percentage.

  Precision=number of target software components retrieved /number of software components retrieved.

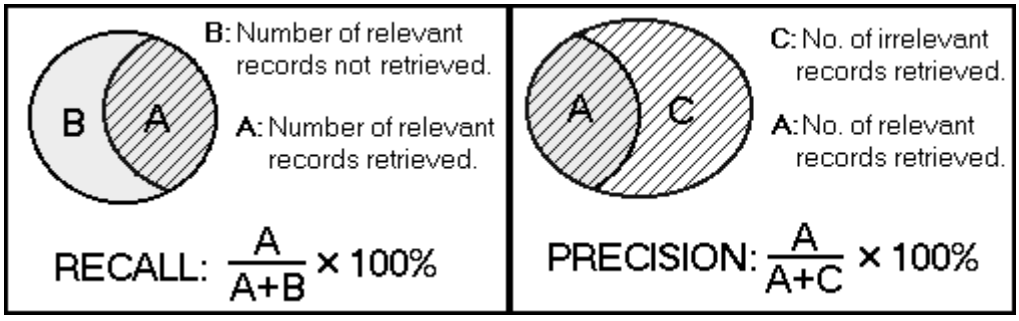


Figure 2.2: Precision and Recall

2.12 URI and Tag URI Standard

For folksonomy to work, the location of resources should be specified. URI is considered to be the most flexible scheme of addressing of resources. According to the URI standard [21], *http://www.tiet.ac.in/syllabus/index.html* -- is a URI, and it has several component parts:

- A scheme name (`http`)
- A domain name (www.tiet.ac.in)

- A path (`/syllabus/index.html`)

A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provides a means of locating the resource by describing its network "location". The term "Uniform Resource Name" (URN) has been used historically to refer to both URIs under the "urn" scheme [RFC2141], which are required to remain globally unique even when the resource ceases to exist or becomes unavailable, and to any other URI with the properties of a name.

2.12.1 TAG URI [RFC 4151]

The *tag* algorithm [22] lets people create identifiers that no one else using the same algorithm could ever create. It is simple enough to do in mind, and the resulting identifiers can be easy to read, write, and remember.

A particular tag URI [23] does not denote a standard based representation but the designer of the system need to look at the particular protocol in which they are used. Tags constitute only a scheme for forming identifiers. There is no authoritative resolution mechanism.

There are reasons for using URI scheme instead of URL scheme in this work:

1. The 'http' might be taken to imply the existence of a web resource that the asset/entity doesn't necessarily want to provide over HTTP.
2. What if the particular company/team doesn't have a domain but they have an email address and they want to identify some resources.
3. Tags are unique for all time by construction. But a domain name (in an http URL) has no long-term guarantees about unique use — it could be re-assigned to an independent party.

CHAPTER 3 Related Work

Two recent trends [23] have brought reuse back to the limelight: open source development projects such as Linux, and component-based software development. If we observe well, the success of open source development rests on individuals contributing code fragments, scripts, and ideas to the public knowledge space through sourcforge or other open source websites associated with the open source projects.

The most likely users of open source knowledge and code are likely to be rookies/freshers rather than experts and temporary project teams rather than the permanent open source community directly associated with the project

However, the success of open source projects hinges on experts and veteran programmers contributing to the public knowledge space—not novices reusing from it.

The success of component-based software requires potential customers reusing from a semi-public knowledge space. Our findings suggest that novices and temporary project teams are the most likely adopters of such software. It is such users and projects to which component-based software should be targeted.The observation was that traditional reuse works better among novice than expert developers, and in transient rather than permanent teams.

3.1 Methods Suggested to Improve Reuse

There are many ideas[10] that has been proposed to improve software reuse :

- Focus on achieving black-box reuse.
- Establish reuse oriented process and organization
- Adopt an incremental approach -employing carefully chosen pilot projects and real life projects.
- Strong foundations for software component composition
- Change adaptable components
- More effective reuse incentive structure

3.2 Related work in Collaborative Tagging

Then we looked at a new way of organizing huge amount of data.Folksonomies are very interesting which can support browsing that can be useful for the end user. But there are observations that folksonomy don't necessarily come up with the keywords related to an artifact. Even it's very difficult for these systems to come up with synonym clusters. Sometimes Folksonomy are believed to have limitations of pure flat lists.

Previous work has proved that they are better than controlled vocabularies in some cases only because controlled vocabularies are not extensible and appealing to larger portion of the population.

Building, maintaining, and enforcing rules of a controlled vocabulary is, relative to folksonomies is very expensive in development time, and in the cost to the user, especially the amateur user or fresher who are new in development, in using the reuse system.

Several authors have documented their thoughts on collaborative tagging but few have done so via research papers. Discussion of collaborative tagging has instead been most active with in blogging community and I have taken most of my content from blogs only. Vander Wal [34] and Mathes [35] have discussed the potential benefits of tagging for personal information management. Vander Wal [34] has observed that in tagging systems there exists a powerful tool, allowing users to index their information resources with their own keywords. But the discussion has concentrated on the use of collaborative tagging for general resource discovery and knowledge organization on the Web or at Enterprise level, much of which has been abstract in nature.

Shirky ([36][37][38]) has suggested that the rise of collaborative tagging on the Web is a 'forced move' and stated the hypothesis that tagging will soon supersede controlled vocabularies for the purposes of resource discovery and knowledge organization. He base his work that current schemes are incapable of reflecting the transient nature of knowledge and therefore the demands of the modern information user. Shirky suggests that collaborative tagging is inclusive; there is no authority imposing a controlled top-down view as we saw in classic ontology based classification of knowledge. All users can participate and contribute their own personal vocabularies (keywords) to generate a

collaboratively built ‘bottom-up’ system which more accurately reflects users’ thinking of the world around them.

3.2.1 Economic Advantages of Collaborative Tagging

The perceived economic advantages of collaborative tagging have also been noted by Shirky [36; 37]. He suggests that the economic advantages will increase in practice and make it the preferred strategy for service providers (for ex: specialized service, in our case, software reuse) and users in the future. The potential cost reductions available by encouraging communities to undertake indexing themselves, as opposed to dependent on professional cataloguers, contributes to the appeal of collaborative tagging.Shirky ([36]; [38]).[39]) has argued that any economies achieved in indexing or classifying resources are simply moved onto the price of resource discovery for users, since the lack of controlled classification increases the number of locations that users have to explore before satisfying their information need.

3.2.2 Clustering in Collaborative Tagging

Davis states that the purpose of controlled vocabularies has not changed and notes that high costs have always been incurred by a very small number of information professionals in order to reduce the discovery costs for a large number of users. Merholz [39] has worked by providing some examples from Connotea .He has found that a query on the subject of ‘Avian Flu’, for example, exposes twenty six terms that have been used to describe essentially the same concept. He concludes that the lexical confusion inherent in tagging should be permitted since it is through this property that a true representation of knowledge is derived. While cataloguers will attempt to keep similar or related concepts together, Shirky argues that it is impossible to cluster such terms without loosing the essence of what each term denotes. He therefore states that it is impossible to disintegrate terms such as ‘software’, ‘open source’ or ‘java’ since their meanings are very different and clustering them together is to lose their concept.

3.3 Search systems

Search systems offer tools for user to search through the assets that repository have indexed. Apart from traditional methods of search like direct keyword based in Google, there are different approaches of search in collaborative systems.

3.3.1 Categorization Browsing in Collaborative Systems

When people were offered search and categorization side by side, fewer and fewer people were using categorization for finding artifacts. The more we scale the harder it becomes to handle the expense of starting a cataloguing system and the problem of maintaining it. The concept of collaborative tagging gained ground when it started from del.icio.us[45].This service was for people who want to track their URLs for themselves but are willing to share their private libraries with each other. This creates an aggregate view of all user's bookmarks.

3.3.2 Improved Search in Collaborative Systems

Mathes [35] and Quintarelli [40] have also found that collaborative tagging can prove beneficial for the ways users search things, providing an increased number of entry points rather than one text box and few options type of interface which is difficult to implement using controlled vocabularies. Mathes finds that the intuitive nature of collaborative tagging, although not necessarily conducive to search-item retrieval or browsing, complements non-search-directed searching and browsing by introducing the user to potentially invaluable resources (components) that would otherwise have been undiscoverable. The psychology of tagging processes experienced by users of a collaborative tagging system has been explored by Sinha [41]. She argues that collaborative tagging utilizes existing natural thinking processes without adding to the mental load experienced by the user. She proposes a cognitive model of the tagging process and highlights the ability of immediate tagging feedback to prevent something she calls as ‘post activation analysis paralyses. According to this concept , such a condition places the user in a state of confusion. Sinha suggests that collaborative tagging reduces the cognitive load experienced by the user because the intellectually difficult task of deciding how a particular resource should be tagged is removed by using system feedback and by observing how others have tagged similar items

3.3.3 Information Retrieval Methods in Collaborative Tagging

Like any other form of metadata, tags are potential answer for information retrieval systems. So far the contribution of IR to folksonomies is mostly a matter of speculation, although Flickr(a tag based online photo solution service) does include a "related tags"

feature which appears to be based on a technique more sophisticated than raw terms counts. We do not yet know whether tags can be usefully clustered or disambiguated through aggregation, nor whether tags would be a useful addition to the data considered by ordinary web search engines. If the tags, taggers and URIs are not sufficient information with which to apply IR methods, then the content of the tagged items and the taggers' social networks are also available. All of those possibilities make the topic complex enough that we may not see clear answers soon.

3.4 Work Done by Open Source Community in Software Repositories

Then we also looked into the work done by open source community to increase the level of trust for any particular software asset while acquiring it from web.BRR (business readiness review) [42] is the project started by CMU that will give companies a trusted, unbiased source for determining whether the open source software they are considering is mature enough to adopt. There are over 100,000 open source projects listed on SourceForge, CodeHaus, Tigris, Java.net and Open Symphony. Some widely adopted projects have become high-quality software suitable for mission critical production environments. Many others are less mature and pose potential risks. Today, companies evaluate open source suitability based on their assessment methods without access to useful assessment data or methods.

The ultimate goal of BRR is to give companies a trusted, unbiased source for determining whether the open source software they are considering is mature enough to adopt. It will help adopters assess which open source software is best suited to their needs and enable them to share findings with the community. It promotes use and adoption of open source software and may assist developers in creating and delivering software geared to enterprise use.

3.5 Collaborative Tagging in Organization

Till date, collaborative tagging is implemented in Internet only. But there is little work done which explores the possibilities of tagging in organization’s Intranet through various applications. We look into some of the work done to introduce collaborative tagging in enterprise. Building an enterprise based [43] software reuse system based on collaborative tagging requires:-

1. Real identies of people that will help the system to find experts on given domain
2. It should be possible to browse the enterprise software library collection anonymously
3. A group may want to create a collection of information sources in support of a project that may be extremely confidential or bound by contract to restricted access. Teams may want to form shared bookmarks that are visible only to the group.
4. We also see a need for private bookmarks.
5. There should be RSS subscription available on every interface page which would supply the latest updates on particular domain.
6. Users may subscribe to a bookmark collection belonging to a particular individual, or to a particular tag or keyword.

Chapter 4 Proposed system and implementation

Based on the issues and trends discussed in chapter 3 of this dissertation, we tried to build a model, which creates the repository based entirely on user feedback and personal utility. A system, which is easy to use, and the low cost of building the repository is the motivation of designing a new system.

4.1 Problem Statement

To apply the concept of collaborative tagging to build a repository, which will help us to search and retrieve software assets, based on tags i.e. a form of keywords assigned by users of the system. This is a bottom up approach of building a repository system in which users assign keywords or tags to the assets they found useful and add to their personal library for future recall. The proposed system also serves as the long tail i.e. addition of uncommon search words by introducing minority keywords and removes the restriction placed on the content of metadata by traditional controlled systems. Recently, systems based on collaborative tagging like del.icio.us (online bookmark repository) and flickr (personal photo database) are gaining rapid popularity on Internet. Del.icio.us has more 300,000 users with minimum performance issues. Based on privacy level set by the user assets can be stored in Public mode or shared with a team or remains private to the user. A successful implementation of proposed solution will lead to potential integration with other software reuse efforts within the organization.

4.2 Overview of the proposed system

The proposed system is a web-based service built on the open source technologies like PHP/MySql. This system helps to tag software assets on different locations and identities with user defined keywords. Once stored in the repository, the component can be browsed and searched through the interface. Users can set the privacy level of the system also while adding their asset to the system. Theoretically all software assets used in software development process can be tagged in this system. The basic purpose is to create a very reliable system that provides privacy, fast retrieval and ease of use.The proposed system codenamed Ksearch basically is represented through a block diagram in

Figure4.1. There can be many users who know the location of the assets found through Internet, Intranet or Ksearch itself. Users will interact with Ksearch and tag the assets with their keywords and adding it to the repository. Now the assets also become the probable result for further search and retrieval. The block diagram describes the same thing.

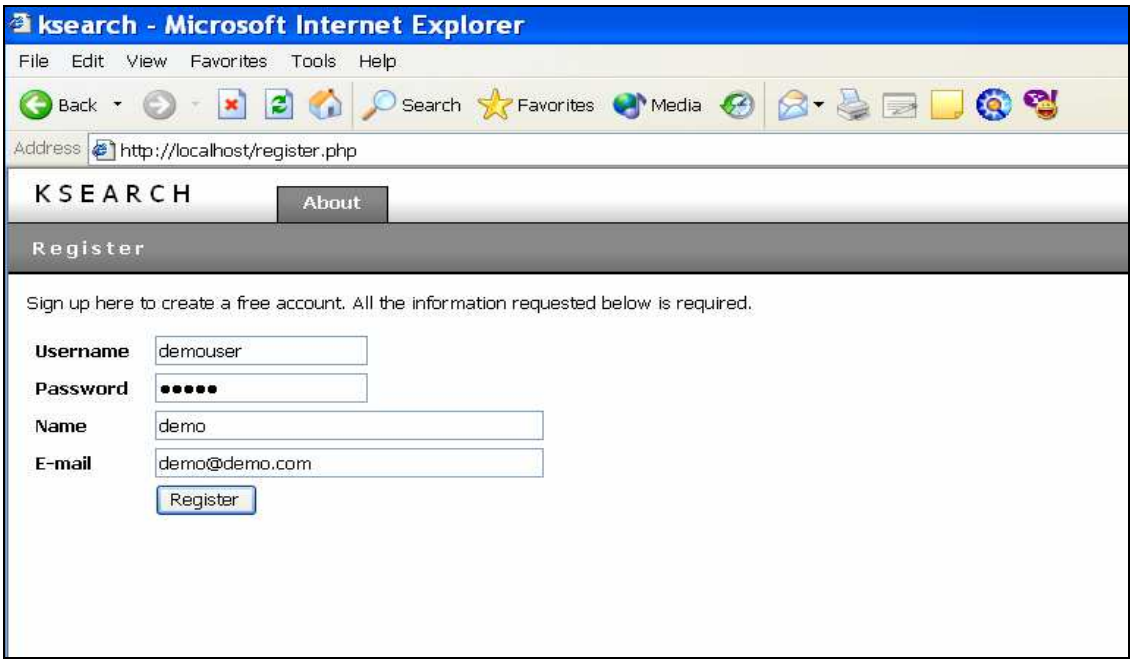
Figure 4.1 Block Diagram of K-search

4.3 Implementation

4.2 Main page of Ksearch

The main goal of this research is to show that collaborative tagging can be integrated to software reuse. We conceptualized a system known as KSearch shown in figure 4.2,

which helps in creating the software repository based on collaborative tagging. In order to use the system, the user has to register to the service through filling a small form as shown in Fig 4.3.This data will be used to identify the user who is using Ksearch and also to maintain his personal



4.3 Registering at Ksearch

4.3.1 Adding Artifacts

Once registered the user can add artifacts to the repository. For that he has to click "add artifact" button. A form comes up for adding assets as shown in Figure 4.4.

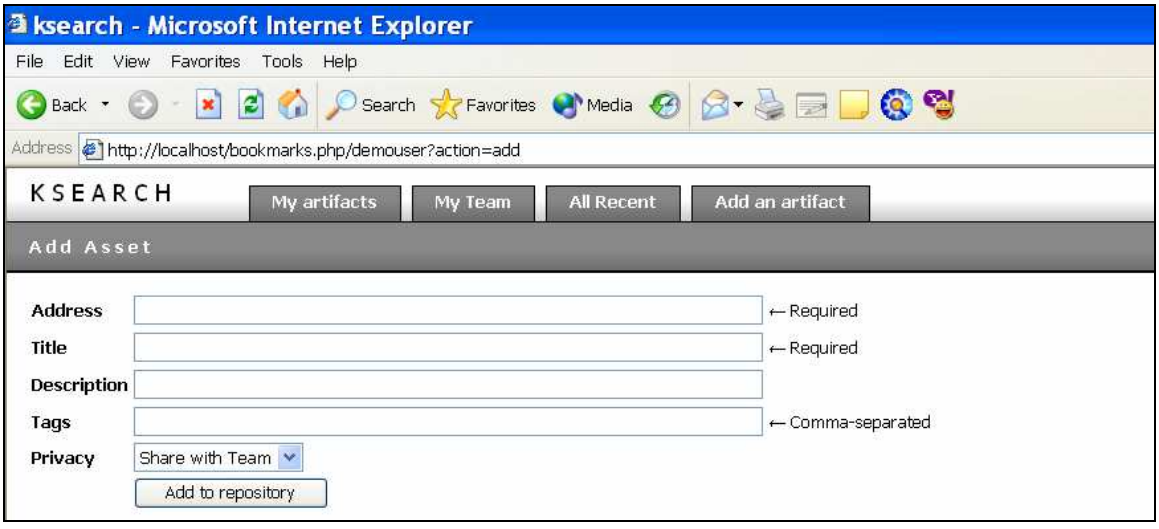


Figure 4.4: Adding component

Descriptions of the fields of add asset form is given below:

- **Address field:**

In Address field, the user is expected to use tag or a URI. The following example URIs illustrates several URI schemes and variations in their common syntax:

- ftp://tiet.ac.in/reuse/javmouse
- http://tiet.ac.in/reuse/javmouse
- ldap://[2001:db8::7]/c=GB?javmouse
- mailto:kapil.bhatia@tiet.ac.in
- news:comp.reuse.www.service.ksearch
- tel:+1-816-555-1212
- telnet://192.0.34.16:80/
- urn:oasis:names: specification:docbook:dtd:xml:4.1.2

Apart from that, if user can tag the item through custom built tag URI. In order to use this scheme this is published as RFC 4151.

We have a component named javmouse, which is a not a obvious name, but we can't be sure it's the only component on the planet with that name. We want to be able to talk about this component using just its name without reference to myself etc and want to be sure, people will not accidentally think we are talking about some other component also named javmouse. So we are going to give him a tag URI.Following are the steps for assigning tag URI :

Step 1. Identify myself: we have two choices: we can use one of e-mail addresses (kapil1312@yahoo.co.in, kapil1312@tiet.ac.in) or we can use a domain name assigned to me (such as kapilbhatia.com). We could also use a shared domain name (tiet.ac.in) if we had explicit permission from the domain holder.

Step 2. Pick a date: It's possible that in 100 years someone else will be using "kapil1312@yahoo.co.in" for e-mail. He may even have a component named javmouse, and I still want my tag to name my javmouse, not his. So we pick some date during which

the address "kapil1312@yahoo.co.in" was definitely mine. We'll pick any date, Monday, May 22,2005.

Step 3. Encode the date as characters, using ISO 8601: "2006-05-22". If we had picked the first day of a month, back in step 2, We would not include the day. If we had picked the first day of a year, we would not include the month or day.

Step 4. Pick a unique name for the object: But it only has to be unique for the already-chosen identity and date. "Javmouse" seems like a fine choice here. We don't want to use a name like "java", because then we may get confused and accidentally call my other java component “java".

Step 5. Combine them:

Combine all the above parts like the format given below:

Tag:kapil1312@yahoo.co.in,2006-05-22:javmouse

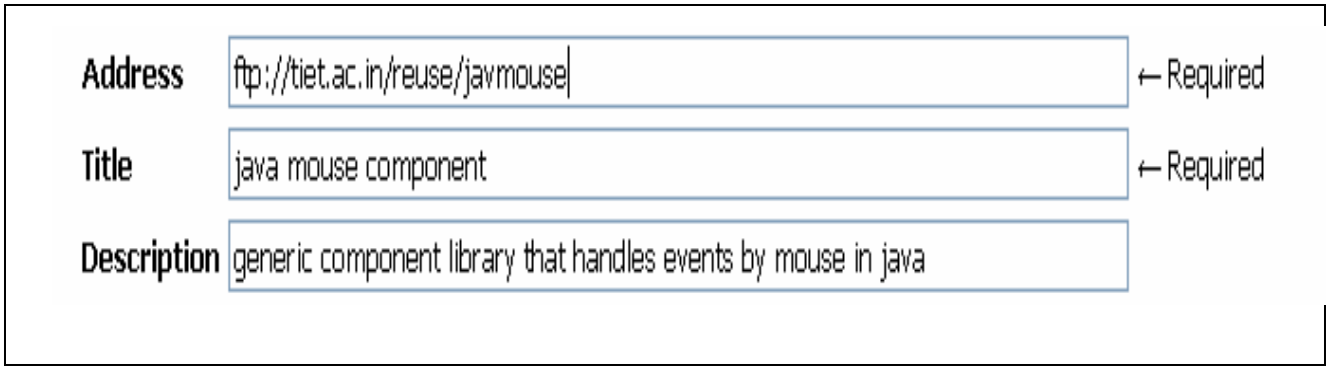


Figure 4.5 Adding a Component

- **Title Field**

Then the user puts the title of the component, which he thinks is suitable for the asset. This field is compulsory so that any other user should be able to know about the asset in every search result he makes.

- **Description Field**

The description field is an optional short note about the asset. This field is not made compulsory as user don’t have time and patience to add so much information for an asset they are adding to their library.

- **Tags Field**

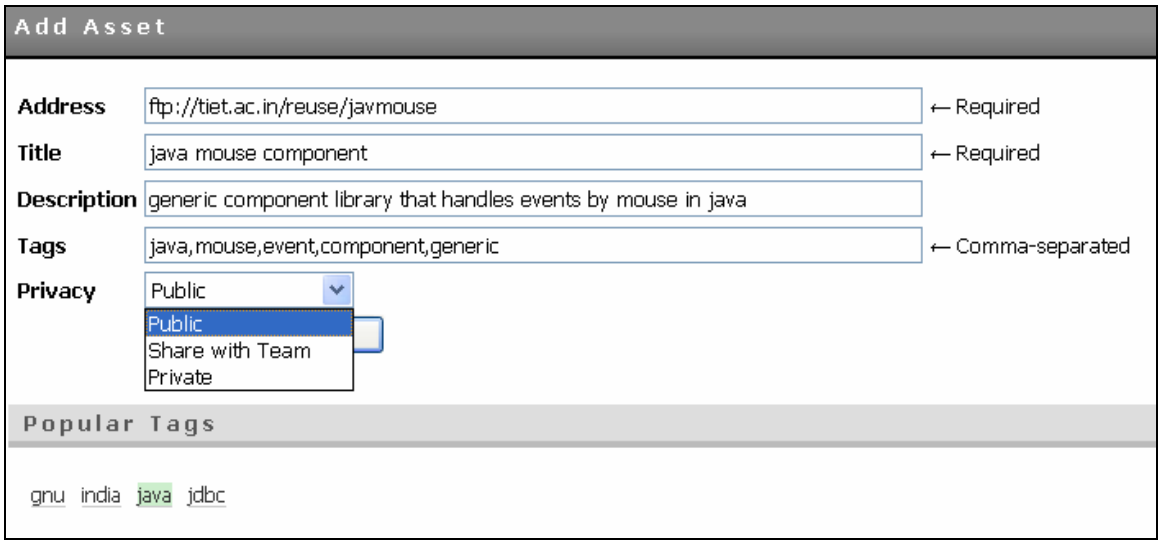


Figure 4.6 Assigning tags and checking privacy

Once writing the description of the components, the component is tagged. Now tagging means assigning keywords. Users can assign tag by themselves or can use the system feedback system by choosing tags from popular keywords as shown in Figure 4.6.Popular tags such as java is selected in this case.

- **Privacy field**

Privacy level for a particular user is set according to which he wants to share this asset with everyone. It can be set as: -

1. Public –The asset and the person who has stored is publicly available in the domain

2. Private-The asset is only available and visible to the person who has stored the asset.

3. With team members- User can add other team members into their watch list so that they can look into the assets acquired by them. Assigning tags and checking privacy is shown in Figure 4.6. It also helps them to readily use those people assets, as level of trust is higher in teams.

4.3.2 Searching Assets

We can search public assets of a particular user. As in Figure 4.7, we can search

1. A user's Flash line (username) public assets
2. Personal library
3. Team member artifacts
4. Anybody Public artifacts

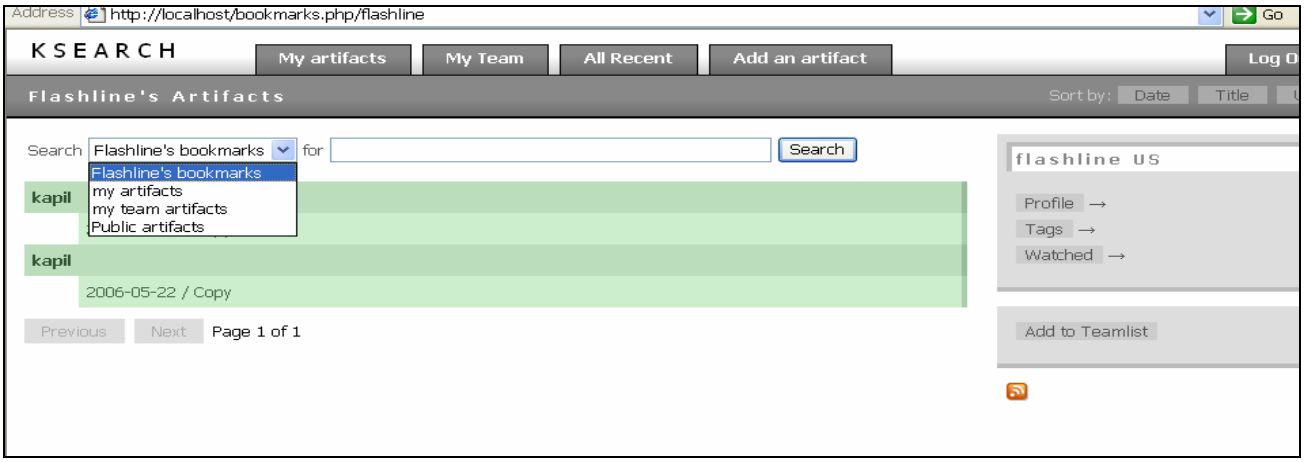


Figure 4.7 Searching Assets

It is very simple to distinguish different assets as shown in Figure 4.8

- All public assets are in light green color
- All team assets are in light red color.
- All personal assets are in pink color.

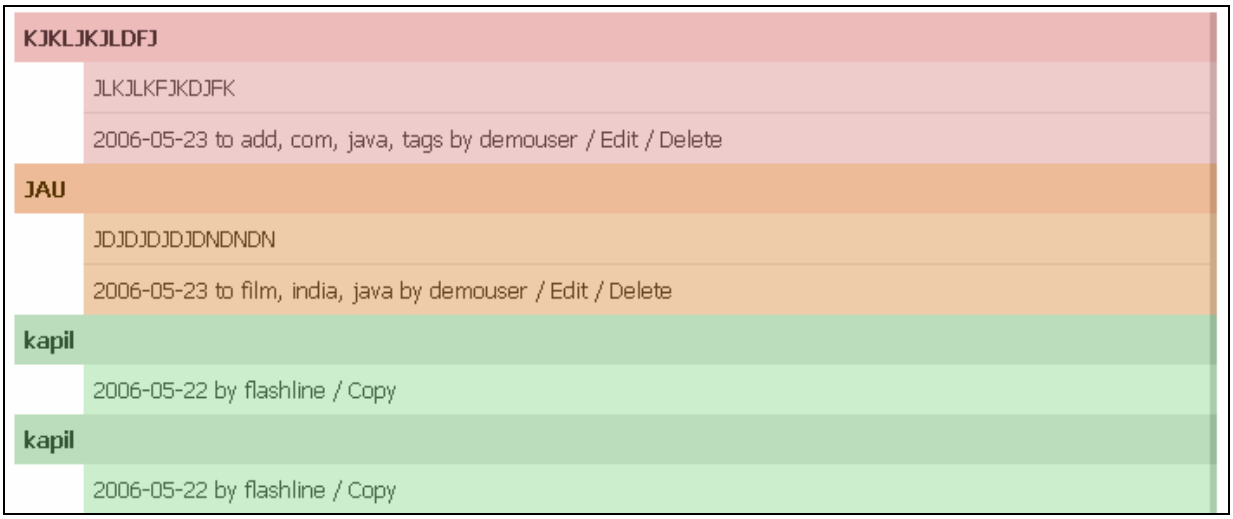


Figure 4.8 Distinguish Different Assets

4.3.3 Browsing

Not just searching but browsing is also allowed in this system. The system finds the most popular tags (as shown in Figure 4.9) assigned by most number of users and the new user can click the tags and browse through to find the software asset he is looking for.

Figure 4.9: Popular Tags Window

As we see in this Figure 4.10, we clicked java and we found all the keywords which have been used with java. In this way we can continue browsing until we find the component we are looking for.

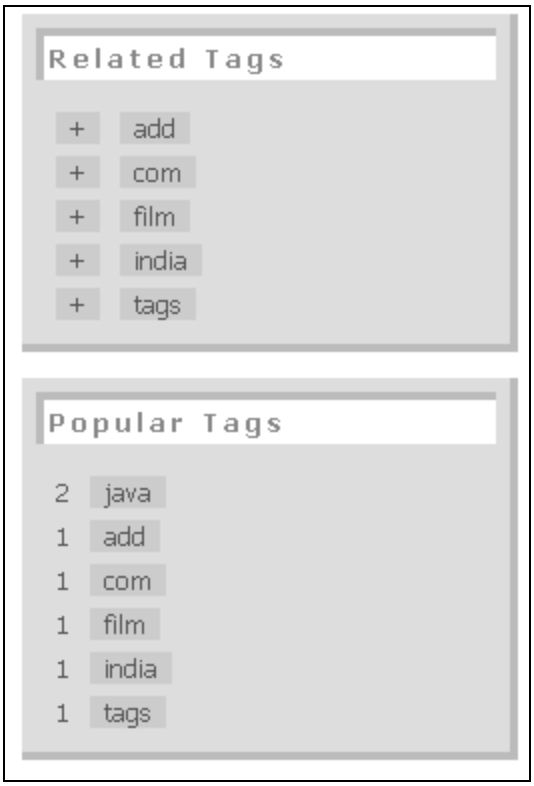


Figure 4.10: Related Tags

4.4 RSS feeds

This system also provides the RSS feeds of the particular user/tag so that the user can update him with the feeds in his personal feed reader. Real Simple Syndication or Rich

Site Summary is an XML format for distributing information about a dynamic web site. Commonly employed by bloggers and news organizations to syndicate new content, allowing users to subscribe to 'RSS feeds' which are usually collected with an 'RSS Aggregator.' In distributed classification systems (such as Ksearch), RSS feeds can be created for particular tags or users.

4.5 Experiments and Results

4.5.1 Setting

The study took place online on the system. Additionally, we gathered data through informal interviews, email and, in face-to-face conversations.

4.5.2 Actors

We collected data from students of software engineering class at TIET, Patiala. These users were given no training whatsoever in using the system so to simulate real world environment.

4.5.3 Events and Procedure

There was a form in which there were three different software assets.(Appendix I)

1. **Code Module:**

   This module in java basically provides how to connect to a database by registering. The JDBC driver using the DriverManager class.

2. **Functional Requirements:**

   This was the functional requirement of admit student procedure which would determine admission certification.

3. **Test Cases:**

   Then there was a test case module, which checks the file in the file path.

Users were given some specified time (of approx. 20 minutes) so that they can understand the code and tag them with their keywords.

4.6 Data Analysis Procedures

These are some of the measures, both quantitative and qualitative:

Quantitative:

- Total number of users in the system
- Total number of items submitted by user (three in our case)
- Number of Tags assigned by user (i.e., what keywords they used)

CASE 1: Semantic Association

With just 12 users adding the three artifacts to their respective personal library, there was an exponential increase in the number and quality of keywords associated with these artifacts. Keywords are shown as tag clouds in collaborative tagging where the most popular tags are in bigger font size..

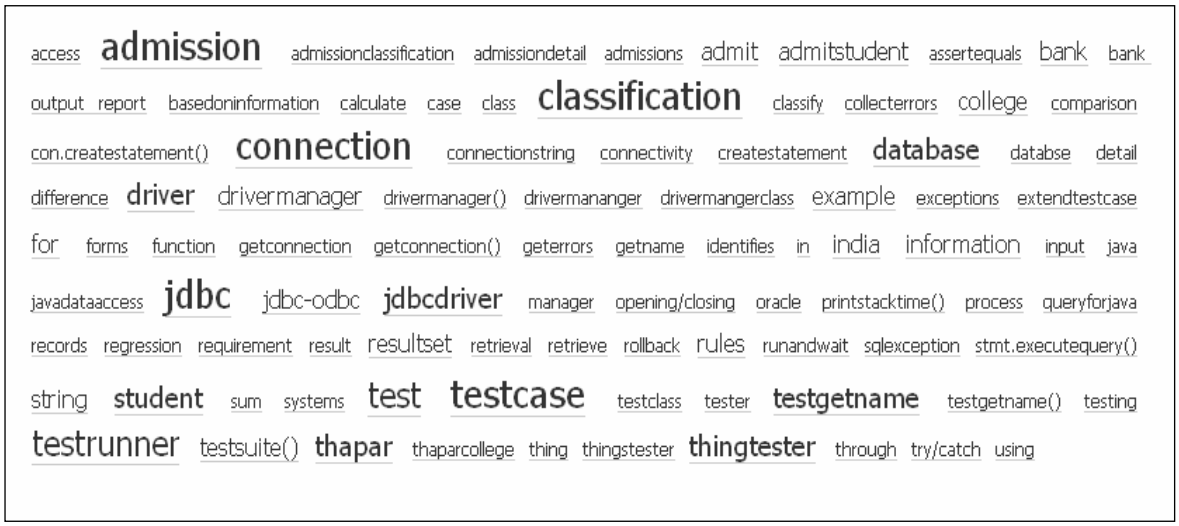


Figure 4.11 Tag Cloud (after collecting data)

Different users considered around 80 keywords appropriate. Now new users can search these artifacts from any of these 80 keywords which resulted in better semantic association and increased efficiency of the system which was lacking in previous reuse systems with such a little cost.

CASE 2: Power Law

Power law tells us about the trust of a particular tag associated with software asset. Power law says that the most used tags are highly visible and are likely to be used by other users. As we see in previous figure more relevant keywords are used again and again which becomes more visible in the system like admission and classification through bigger font. Instead of creating all tags from scratch, users tend to take tags from the popular tags .This helps in classifying an artifact through some limited number of good keywords, which has been identified by the broad consensus of the users.For example, in Figure 4.12, user has used popular tags like admission and admitstudent which has been used by many users.

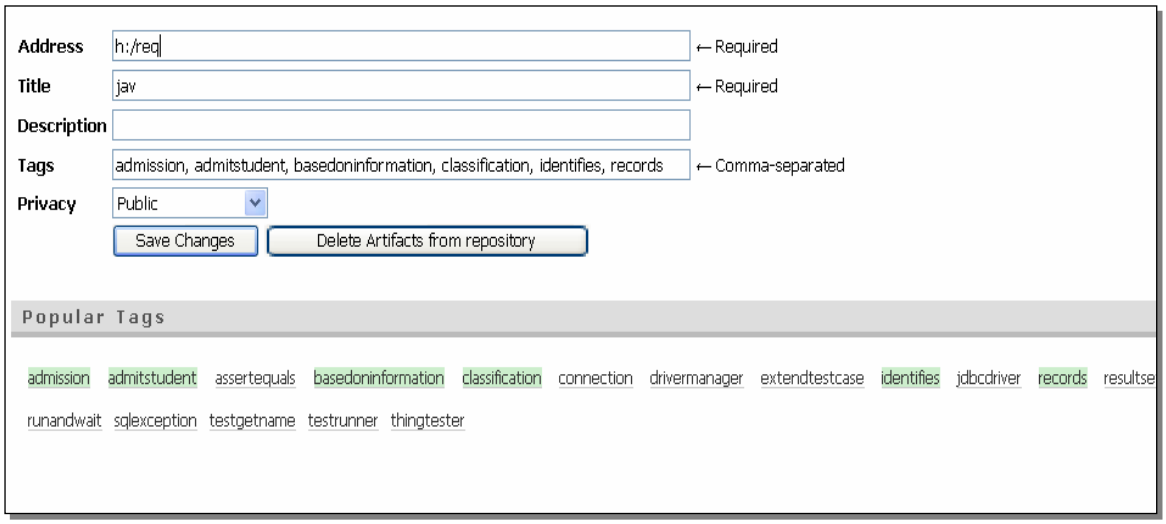


Figure 4.12 Power Law

CASE 3: Experience Based User Learning

The system tends to self-learn with more users using it. For example if a user search for a component “binary tree creation”. Now in user’s mind, its “make binary tree” rather than “create binary tree”. As through initial search “make binary tree”, user will not get any result and he will more likely tend to generalize the search to just “binary tree”.

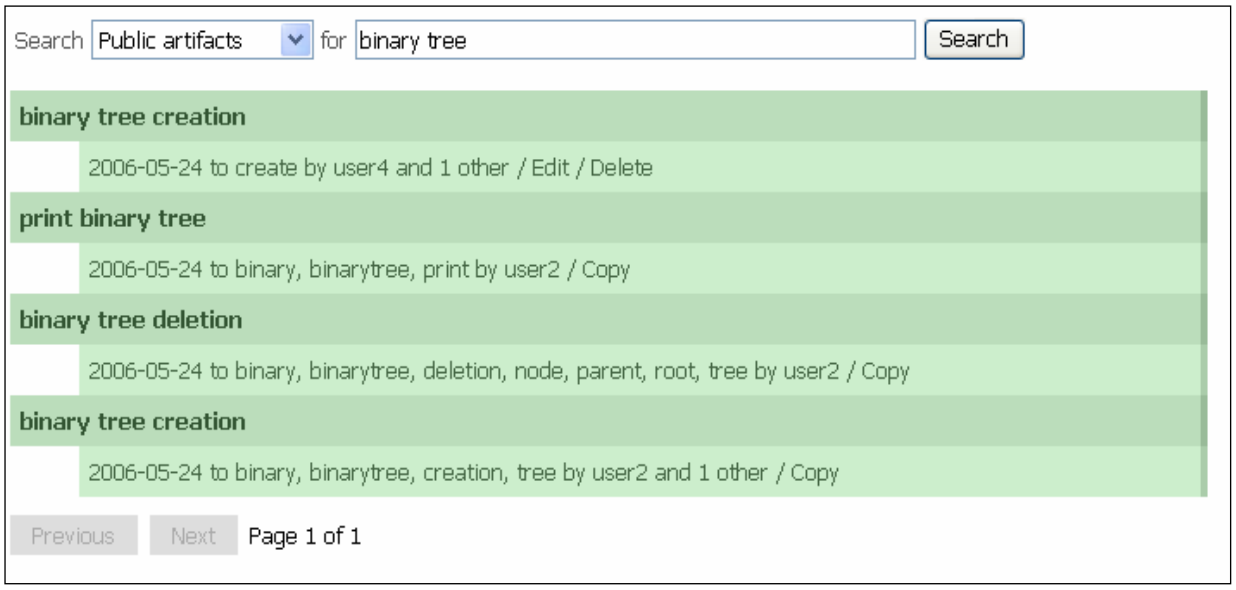


Figure 4.13 Searching Binary Tree

As the user look into the search result, he will most likely click on binary tree creation, which according to him is "binary tree make".

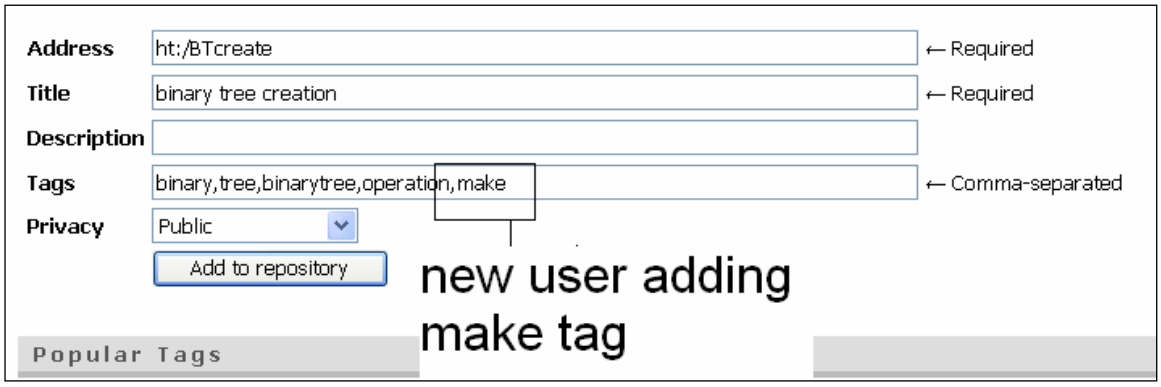


Figure 4.14 Making system learn

So next time if someone use "make" keyword to search component, he will get the results as shown in fig 4.15.System does self-learning as users make their personal libraries more efficient.

Fig 4.15 Optimized Library

CASE 4: Wrong Tagging

Incase the user puts the wrong tag in some artifact, the efficiency of the system decreases. As there is no verification by the system on the quality or quantity of the tag. For example the user can use "delete" tag for binary tree creation.

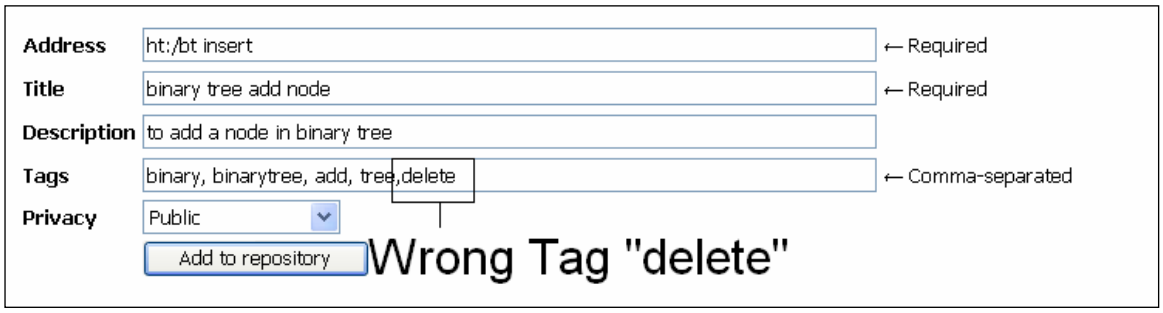


Figure 4.16 Assigning Wrong Tag

The use of wrong tags can be due to lack of knowledge of the user of that component or the bad intentions of the user. But as more people judge a component by the more popular tags it uses and add their own tags, the effect of wrong tags gets depreciated with more and more people using the system and accessing the tag.

CASE 5: Sharing and Security

User can set the sharing limits of a particular asset with a team or make it public or just confine it to himself. As found in previous work that reuse is more likely happen in transient team rather than permanent teams, this system facilitates reuse at team level. User can add or remove any other user of the system in his team list as shown in Figure 4.17.

Figure 4.17 Add users to team list

As team members trust each other and understand each other style of coding or working, reuse at this level is fast and can increase the level of reuse at organization level.

CASE 6: Scalability

Scalability of collaborative systems in terms of number of tags and keywords is tested on other systems by various authors with systems like delicious. We are referring to the performance value of Scuttle and other collaborative tagging systems as Ksearch is based on Scuttle, a collaborative tagging based system.

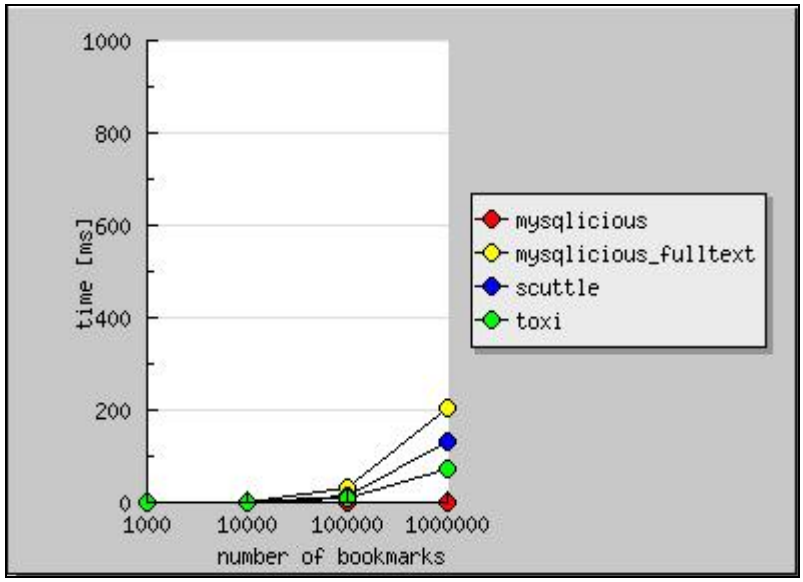


Figure 4.18-Tag Systems Performance

The comparison of performance of different tag systems i.e. mysqlicious, toxi[44] etc is calculated with number of items ranging from 1000 to 100,000 and we found that this system works well upto million items. Above that limit, the system performance degrades.

Other observations regarding the quality of tags:

1. **Misspelt Tags –** misspelt tags such as admision or admition
2. **Compound Word Grouping –**As users of the system were not trained or educated, they used tags like admit student, based on information, collect errors.Users were more comfortable grouping names as used in functions as they are easier to tag. But splitting the function names was almost NIL in code tagging.
3. **Singular/Plural –** there were some tags which were used in both singular and plural form like rule and rules.
4. **Use of special symbols –**
    i. Use of hyphen: like hyphen in tags like jdbc-odbc
    ii. Use of brackets: usually used in name of function like printstacktime (), test suite ().
    iii. Use of dot operator: used in class.functionname kind of keywords like stmt.executequery.
    iv. Use of forward or backward slash: Few users have used slash to combine two keywords in a tag like try/catch.
5. **Generic Tags-** Few people have tagged items with keywords which are very general in nature which can't be used for any purpose like through, like, thing etc.

Chapter 5 Conclusion and Future Work

Developing quality software with less cost is difficult in today scenario. Software reuse seems to be one of the solutions of this ever-lasting problem. Using assets developed for one application program in another application program is the most simplistic view of reuse. There are other views and advantages of software reuse as well, one of which we have tried to explore in our work i.e. building repositories on collaborative tagging.

5.1 Conclusion

Now we conclude our work with following results:

- The implemented system considers the mental perception of the user and stores the location of asset in the repository along with the user knowledge or experiences. The main problem in construction of any effective reuse repository is the representation of the component in the repository. This system provides an alternative solution to this challenge by using user-defined specification of assets. User specifies the tags for components and these tags will represent the metadata of the component which are used for indexing and retrieval of components.
- Users need not specify the components according to predefined structure and classification scheme of the repository .In our proposed system, users can simply specify the structure and classification scheme of the repository according to their keywords while using the system.
- Experimental results show that the system results in better semantic association, which would help in better subsequent search as demonstrated in Section 4.6, Case1.
- Importance of personal and general consensus among people regarding keywords associated with assets is given its due weightage in the system as demonstrated in Section 4.6, Case 2.
- The system becomes self-learning as more people use the system and gives their knowledge to the system to increase its vocabulary and ultimately its capability to

return components that are needed to solve their problems as given in Section 4.6 Case 3.

- Wrong tags assigned to the assets tend to decrease the efficiency of the system as shown in Section 4.3 Case 4 but the problem is reduced as more people tag that asset with correct keywords.
- As sharing among trusted members or among teams is the big thrust to software reuse, this system facilitates resource sharing at three levels (personal, team, public) and as a result the proposed system becomes very useful for the organization to build high quality software products and services as shown in Section 4.6 case 5.

5.2 Future Work

5.2.1 Integration of already existing tools with Collaborative Tagging

If information retrieval systems begin to incorporate user-centered information management tools like the proposed system, the organizational knowledge developed by the users have the possibility to be of great interest to other users and improve the systems.

5.2.2 Auto Tagging

In future, we can have applications in which user and the system add tags to the system. The machine tags can be generated by mapping the user thinking process and the keywords user considers when assigning metadata to previous components.

5.2.3 Hierarchal Multi Labeling

Most collaborative tagging services support single or plain level of tagging as objects should fall under multiple categories. It can be done by representing labels in trees or another data structures.

5.2.4 Building a tagging Interaction model

A better approach will be to build tags into the interaction model of applications, turn the actions and intentions of users into inferred or implied tags, then re-surface that information as a basis for explicit meta-tagging action later.

References

[1] Charles W. Krueger, ACM computing surveys, volume 24, Issue 2 (June 1992) , Pp: 131 - 183

[2] Will Tracz,ACM SIGSOFT Software Engineering Notes, Volume 13 , Issue 1 (January 1988) ,Pp.: 17 - 21

[3] William B. Frakes, Kyo Kang, Software Reuse Research: Status and Future, IEEE transaction on software engineering (July 2001),pp.: 529-536

[4] Kevin C Desouza, Yukika Awazu and Amrit Tiwana, Four Dynamics for Bringing Use Back into Software Reuse published in the Communications of the ACM,(January 2006).

[5] T.Ravichandran,Marcus A.Rothenberger, Software reuse strategies and component markets, Communications of the ACM, Volume 46 , Issue 8 (August 2003),Pp.:109-114

[6] Methods to acquire software components,Flashline available at http://www.flashline.com/docs/WPGRISS02.pdf,http://www.flashline.com/docs/WPHIGHSMITH01.pdf

[7] Mili, A. Chmiel, S.F. Gottumukkala, R. Zhang, L. West Virginia Univ., Morgantown, WV, An integrated cost model for software reuse, Proceedings of the 22nd international conference on Software engineering, Ireland, 2000, pp. 157-166

[8] Tutorial on Component based development available at http://www.webcabcomponents.com/componentization.shtml

[9] Will Tracz, International Conference on Software Engineering, Proceedings of the 16th international conference on Software engineering, Sorrento, Italy,Pp.: 271 - 272

[10] Managing software productivity and reuse Boehm, B. ,Univ. of Southern California, Los Angeles, CA; September (1999) pp. 111-113

[11] Duval, E., Hodgins, W., Sutton, S. & Weibel, S, L. (2002), "Metadata Principles and Practicalities", DLib Magazine, Vol. 8 No. 4, available at: http://www.dlib.org/dlib/april02/weibel/04weibel.html

[12] Greenberg, J. "Metadata extraction and harvesting: a comparison of two automatic metadata generation applications", Journal of Internet Cataloging, Vol. 6 No. 4,2004, pp.59-82.

[13] Anderson, J, D. & Perez-Carball, J. , "The nature of indexing: how humans and machines analyze messages and texts for retrieval. Part I: Research, and the nature of human indexing",Information Processing & Management, Vol. 37 No. 2,2001, pp.231-254.

[14] Quintarelli, E. , "Folksonomies: power to the people", Proceedings of the 1st International Society for Knowledge Organization (Italy) (ISKOI), UniMIB Meeting, June 24, Milan, Italy,2004, ISKOI,Italy

[15] David N. Sturtz,Communal Categorization:The Folksonomy, Content representation,December 16, 2004

[16] Computer Mediated Communication - LIS590CMC Graduate School of Library and Information Science University of Illinois Urbana-Champaign

[17] Guy, M. & Tonkin, E. (2006), "Folksonomies: Tidying up Tags?" D-Lib Magazine, Vol. 12 No. 1, available at: http://www.dlib.org/dlib/january06/guy/01guy/html

[18] Adam Mathes,Folksonomies - Cooperative Classification and Communication Through Shared Metadata http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

[19] Benjamin Szekely, Elias Torres,Ranking Bookmarks and Bistros,Intelligent Community and Folksonomy Development,Benjamin Szekely, Elias Torres,Harvard University{bszekely,torres}@fas.harvard.edu,May, 2005

[20] Search guid basics available at http://www.frame.org.uk/Searching for Information/Basics.htm

[21] Tag URI Basics available at www.TagURI.org

[22] RFC 4151-Tag URI available at http://www.faqs.org/rfcs/rfc4151.html

[23] URI(URL syntax)-RFC3986 http://www.ietf.org/rfc/rfc3986.txt

[24] M Morisio, M Ezran, C Tully ,Success and failure factors in software reuse - Software Engineering, IEEE Transactions on, 2002

[25] Software reuse through information retrieval, ACM SIGIR Forum, Volume 21, Issue 1-2 Sept.-March 1986-1987, Pp.: 30 - 36

[26] Ruben Prieto-Diaz ,Implementing Faceted Classification for software Reuse, software Production Consortium, Herndon, VA,ACM Press New York, NY, USA; Pp.: 88 – 97: 1991

[27] Introduction to Faceted classification available at http://www.kmconnection.com/DOC100100.htm

[28] J.M.Wing, A Specifier's Introduction to Formal Methods,Computer,volume 23,Issue 9,Pp. 8,10-22,24

[29] Clustering algorithms, available at :http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/

[30] Masoud Nikravesh , Roy D. Adams , Raymond A. Levey , Soft computing: tools for intelligent reservoir characterization, Berkeley Initiative in Soft Computing BISC , Computer Science Division, Department of EECS, university of California, Berkeley.

[31] Self organizing maps available at: http://davis.wpi.edu/~matt/courses/soms/

[32] Shirky,When does ontological classification work available at http://shirky.com/writings/ontology_overrated.html#when_does_ontological_classification_work

[33] Todd L. Veldhuizen,Software Libraries and the Limits of Reuse: Entropy, Kolmogorov Complexity, and Zipf's Law, Open Systems Laboratory, Indiana University Bloomington,Bloomington, IN, USA.

[34] Vander Wal, T. , "Explaining and Showing Broad and Narrow Folksonomies",Vanderwal.net,http://www.vanderwal.net/random/entrysel.php?blog=1635,2005

[35] Mathes, A. , "Folksonomies – Cooperative Classification and Communication Through Shared Metadata", Adam Mathes.com, USA http://adammathes.com/academic/computermediatedcommunication/folksonomies.pdf, 2004

[36] Shirky, C. (a), "Ontology is Overrated: Categories, Links and Tags", Shirky.com, http://shirky.com/writings/ontology_overrated.html

[37] Shirky, C. (b), "Folksonomies are a forced move: a response to Liz", Many2Many:A group Weblog of social Software http://www.corante.com/many/archives/2005/01/22/folksonomies_are_a_forced_move_a_response_to_ liz.php

[38] Shirky, C. (c), "Semi-Structured Meta-data has a Posse: A response to Gene Smith", You're It! A Blog on Tagging, available: ttp://tagsonomy.com/index.php/semi-structured-meta-data-has-a-posse-aresponse-to-gene-smith

[39] Merholz, P. , "Clay Shirky's Viewpoints are Overrated", Peterme.com: links, thoughts, and essays from Peter Merholz, available at: http://www.peterme.com/archives/000558.html

[40] Quintarelli, E. , "Folksonomies: power to the people", Proceedings of the 1st International Society for Knowledge Organization (Italy) (ISKOI), UniMIB Meeting, June 24, 2005,Milan, Italy, ISKOI, Italy, available at: http://www.iskoi.org/doc/folksonomies.htm

[41] Sinha, R. , "A cognitive analysis of tagging", Rashmi Sinha's weblog, available: http://www.rashmisinha.com/archives/05_09/tagging-cognitive.html

[42] Shirky,When does ontological classification work available at http://shirky.com/writings/ontology_overrated.html#when_does_ontological_classification_work

[43] Social Bookmarking in the Enterprise,ACM Queue vol. 3, no. 9 - November 2005 by David Millen, Jonathan Feinberg, and Bernard Kerr, IBM

[44] Phillip Keller,TagSystems Performance Test available at http://www.pui.ch/phred/archives/2005/06/tagsystems-performance-tests.html

[45] A social Bookmarking site based on collaborative tagging available at http://del.icio.us

Paper/Publication

Kapil Bhatia, Rajesh K. Bhatia, "Collaborative Tagging for Building Software Repositories", communicated to 10th IASTEAD International conference on Internet and Multimedia Systems and Applications, Honolulu, Hawaii, USA, to be held on August 14-16,2006

Appendix I

Artifacts used in the User feedback form for Ksearch

a. **Code**

```
/*
 * $Log: JdbcConnection.java,v $
 * Revision 1.1  2000/11/30 18:13:29  roirex
 * Added for new layout.
 *
 * Revision 1.7  2000/09/16 14:01:49  roirex
 * Now using Log4J
 *
 * Revision 1.6  2000/03/15 19:52:35  y9ms5f
 * Added LogHelper, Taborder + usual bugfixes
 *
 * Revision 1.5  2000/03/11 16:30:40  y9ms5f
 * Many PostgreSQL improvements.
 *
 * Revision 1.4  2000/03/04 21:04:37  roirex
 * Added functions for Installing Airlog. Fix bug with Lock-problem for InstantDB.
 *
 * Revision 1.3  2000/02/26 21:30:21  y9ms5f
 * Added close() to close. Needed by PostgreSQL' driver.
 *
 * Revision 1.2  2000/02/25 21:54:03  roirex
 * Java Doc Added.
 *
 *
 * Copyright (c) 2000 Frederik Hansen.
 * <roirex@post1.tele.dk>
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.          See
the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 * If you modify this file, please send us a copy.
 */
```

```
package dk.highflier.airlog.utility;

import java.sql.*;
import java.util.*;

/**
 * Synopsis:    For handling database querys and updates in a easy way.<BR>
 *
 * License:             GNU GPL2, look for details in file "COPYING"
 *
 * @author              Frederik Hansen, roirex@post1.tele.dk
 *
 */
public class JdbcConnection
{
   public Connection conn;
   Statement stmt = null;
   private org.log4j.Category log =
org.log4j.Category.getInstance("Log.JdbcConnection");

   java.util.Properties info;
   String url;

   /**
    * Method initializes the JdbcConnection.
    *
    * @remarks                  Autocommit is set to false.
    *
    * @param            user, username for the database user.
    * @param            password, password for the database user.
    * @param            prefetch, the default number of prefetch rows.
    * @param            driver, the database JDBC driver.
    * @param            url, the URL for the database.
    */
   public JdbcConnection(String user, String password, int prefetch, String driver, String
url) throws Exception
   {
      log.info("Connecting to " + user);

      try
      {
         Class.forName(driver);

         info = new java.util.Properties();
         info.put("user", user);
         info.put("password", password);
```

```
        info.put("prefetch", String.valueOf(prefetch));
        this.url = url;

        try
        {
            conn = DriverManager.getConnection(url, info);
            conn.setAutoCommit(false);

            log.info("Connected");
        }
        catch( Exception e )
        {
            log.error( "Could not connect to database (" + e.getMessage() + ")");
            throw e;
        }
    }
    catch( ClassNotFoundException e )
    {
        log.error("Could not load the driver ", e);

        throw e;
    }
}

        /**
 * Method creates and returns a Statement.
 */
public Statement getStatement() throws SQLException
{
        Statement stmt = null;

    try
    {
        stmt = conn.createStatement();
        }
        catch (SQLException sqlExcep )
    {
            if ((sqlExcep.getErrorCode() == 0) && (sqlExcep.getSQLState() == null))
            {
                conn = DriverManager.getConnection(url, info);

                log.info("Reconnected");

                return conn.createStatement();
            } else
            {
```

```
                throw sqlExcep;
            }
        }
        catch (Exception e )
        {
            log.debug(e);
        }

        return stmt;
}

/**
 * Method execute an Update in the database.
 * Is usefull if the connection to the database is lost.
 */
private int executeUpdate(String s) throws SQLException
{
    int result = 0;

    try
    {
        Statement stmt1 = conn.createStatement();

        try
        {
            stmt1.execute(s);
            result = stmt1.getUpdateCount();
        }
        catch (SQLException e1)
        {
            stmt1.close();

            throw e1;
        }

        stmt1.close();
    }
    catch (SQLException e)
    {
         throw e;
    }

        return result;
}

/**
```

```
 * Method executes an update-statement in the database.
 * @returns Number of updated rows.
 */
public int execute(String s) throws SQLException
{
   try
   {
      return executeUpdate(s);
   }
   catch (SQLException sqlExcep)
   {
      if ((sqlExcep.getErrorCode() == 0) && (sqlExcep.getSQLState() == null))
      {
         conn = DriverManager.getConnection(url, info);

         log.info("Reconnected");

         return executeUpdate(s);
      } else
      {
         throw sqlExcep;
      }
   }
   catch (NullPointerException nullexcep)
   {
      conn = DriverManager.getConnection(url, info);

      log.info("Reconnected");

      return executeUpdate(s);
   }
}

/**
 * Method creates and returns a PreparedStatement.
 */
public PreparedStatement prepareStatement(String statement) throws SQLException
{
   try
   {
      return conn.prepareStatement(statement);
   }
   catch (SQLException sqlExcep)
   {
      if ((sqlExcep.getErrorCode() == 0) && (sqlExcep.getSQLState() == null))
      {
```

```
                conn = DriverManager.getConnection(url, info);

                log.info("Reconnected");

                return conn.prepareStatement(statement);
            } else
            {
                throw sqlExcep;
            }
        }
        catch (NullPointerException nullexcep)
        {
            conn = DriverManager.getConnection(url, info);

            log.info("Reconnected");

            return conn.prepareStatement(statement);
        }
}

    /**
     * Method commit changes to the Database.
     */
    public void commit()
    {
        try
        {
            conn.commit();
        }
        catch ( SQLException e )
        {
            log.debug(e);
        }
    }

    /**
     * Method rollback changes from the Database.
     */
    public void rollback()
    {
        try
        {
            conn.rollback();
        }
        catch ( SQLException e )
        {
```

```
            log.debug(e);
        }
    }

    /**
     * Method closes connection to Database
     */
    public void close()
    {
        try
        {
            if (conn != null)
                conn.close();
        }
        catch ( SQLException e )
        {
            log.debug(e);
        }
    }
}
```

b.**Functional Requirement**

Process: Admit Student

Process Step: Determine Admission Classification

Functional Requirement name: Calculate Admission Classification

Description: Identifies and records the appropriate classification based on information provided by the student and rules in Thapar College.

c. **Test Case**

```
public class ThingTester extends TestCase
{
                public ThingTester(String name)
                {
                        super(name);
                }
```

```
        public static void main(String[] args)
        {
            TestRunner.runAndWait(newTestSuite(ThingTester.class));
        }

    public void testGetName() throws Exception
        {
            String fileSpec=new String("c:\\xxx\\yyy\\zzz.txt");
            assertEquals("zzz.txt",getName(fileSpec));
        }
}
```